Desarrollo FrontEnd con JavaScript

s01 - Entendiendo JavaScript

Temas

- Semana 01: Entendiendo JavaScript
- Semana 02: Manejo del DOM y APIs del navegador
- Semana 03: Peticiones asíncronas con XmlHttpRequest
- Semana 04: jQuery
- Semana 05: Aplicaciones Web I : Modelos y colecciones
- Semana 06: Aplicaciones Web II : Vistas y routers

Single-page web apps

Web applications (no web sites!)

Desarrollo FrontEnd

HTML5

JavaScript

CSS3

Patrones de diseño

JavaScript != Java

JavaScript != jQuery

JavaScript

JavaScript != FrontEnd*

*(aunque casi)

Sintaxis similar a C/C++, PHP...

Tipado dinámico

JavaScript

Orientada a objetos*

*(No hay clases, hay prototypes)

Las funciones son objetos*

*(Y también son constructores de objetos)

Arrays

- push
- pop
- slice
- splice
- indexOf

```
var numbers = [];
numbers.length;   // 0
numbers;          // []

numbers.push(2);
numbers.length;   // 1
numbers;          // [2]

numbers.push(4);
numbers.length;   // 2
numbers;          // [2, 4]
```

Arrays

- push
- pop
- slice
- splice
- indexOf

```
numbers.push(6);
numbers.length;    // 3
numbers;           // [2, 4, 6]

numbers.pop();     // 6
numbers.length;    // 2
numbers;           // [2, 4]
```

Arrays

- push
- pop
- slice
- splice
- indexOf

```
var numbers = [2, 4, 6, 8];
numbers.slice(1, 3);
// [4, 6]
numbers;
// [2, 4, 6, 8]

var dup = numbers.slice(0);
numbers.pop();
numbers;              // [2, 4, 6]
dup;                  // [2, 4, 6, 8]
```

Arrays

- push
- pop
- slice
- splice
- indexOf

```
var numbers = [2, 4, 6, 8];
numbers.splice(2, 2);
// [6, 8]
numbers;
// [2, 4]
```

Arrays

- push
- pop
- slice
- splice
- indexOf

```
var numbers = [2, 4, 22, 8];
numbers.indexOf(22);
// 2
```

Objetos

- for .. in
- instanceof
- typeof
- hasOwnProperty

```
var book = {
  name: 'JavaScript: The Good Parts',
  author: 'Douglas Crockford',
  price: 29.99,
  pages: 172,
  releaseDate: new Date('2008-5-1')
};
```

Objetos

- for .. in
- instanceof
- typeof
- hasOwnProperty

```
for (attribute in book) {
  console.log(attribute + ' : ' +
      book[attribute]);
}

// name : JavaScript: The Good Parts
// author : Douglas Crockford
// price : 29.99
// pages : 172
// releaseDate : Thu May 01 2008
00:00:00 GMT-0500 (Hora est. Pacífico,
Sudamérica)
```

Objetos

- for .. in
- **instanceof**
- typeof
- hasOwnProperty

```
book.releaseDate instanceof Date;
// true
book.releaseDate instanceof Object;
// true
book.pages instanceof Number;
// false
book.name instanceof String;
// false
```

Objetos

- for .. in
- instanceof
- **typeof**
- hasOwnProperty

```
typeof book.releaseDate;
// "object"
typeof book.pages;
// "number"
typeof book.name;
// "string"
```

Objetos

- for .. in
- instanceof
- typeof
- **hasOwnProperty**

```
book.hasOwnProperty('pages');
// true
book.hasOwnProperty(‘isbn');
// false
```

Funciones

- apply
- call

```
var sumTotal = function(base, price) {
  var total = base + price;

  return total;
};

sumTotal(0.00, 29.99);
// 29.99
```

Funciones

- apply
- call

```
sumTotal(0.00, 29.99);
// 29.99
sumTotal.apply(this, [0.00, 29.99]);
// 29.99
```

Funciones

- apply
- call

```
sumTotal(0.00, 29.99);
// 29.99
sumTotal.call(this, 0.00, 29.99);
// 29.99
```

Funciones – Aceptando “n” parámetros

```
var sumTotal =
function(base, price) {
  var total = base + price;

  return total;
};
```

```
var sumTotal = function() {
  var prices = arguments,
      total = 0;

  for (var i = 1; i <
prices.length; i++) {
    var price = prices[i];
    total = total + price;
  }

  return total;
};
```

Funciones – Usando apply

```
var addToCart = function(user, book) {
  user.cart.push(book);

  var total = [];

  for (var i = 0; i < user.cart.length; i++) {
    var purchasedBook = user.cart[i];
    total.push(purchasedBook.price);
  }

  user.total = sumTotal.apply(this, total);
};
```

Usando objetos y funciones

```
var user = {
  cart: [],
  total: 0
}
```

```
var book1 = {
  name: 'JavaScript:
The Good Parts',
  author: 'Douglas
Crockford',
  price: 29.99,
  pages: 172,
  releaseDate: new
Date('2008-5-1')
};
```

Usando objetos y funciones

```
var book2 = {
  name: 'JavaScript: The Definitive Guide',
  author: 'David Flanagan',
  price: 49.99,
  pages: 1100,
  releaseDate: new Date('2011-4-1')
};
```

```
addToCart(user, book1);
user.total;
// 29.99
addToCart(user, book2);
user.total;
// 79.98
```

Funciones - Constructores

```
var User = function User(attributes) {
  this.id = attributes.id;
  this.email = attributes.email;
  this.nickname = attributes.nickname;
};

var user = new User({
  id: 616,
  email: 'gustavo.leon@outlook.com',
  nickname: 'hpneo'
});
```

Prototype

Extendiendo objetos

Prototype - Extendiendo objetos

- Las funciones reemplazan a las clases al construir objetos.
- Las funciones son objetos que tienen propiedades y métodos, por lo que son llamadas First-Class Citizens.
- La propiedad prototype sirve para extender objetos creados con funciones.

```
var User = function User(attrs)
{
  this.id = attrs.id;
  this.email = attrs.email;
  this.nickname =
attrs.nickname;
};
```

Prototype - Extendiendo objetos

```
User.prototype.setEmail = function(email) {
  this.email = email;
};

User.prototype.getEmail = function() {
  return this.email;
};

user = new User({});
user.setEmail('gustavo.leon@outlook.com');
user.getEmail();
// "gustavo.leon@outlook.com"
```

Prototype - Extendiendo objetos nativos

- Todos los objetos de JavaScript pueden ser extendidos, incluso los objetos nativos o del navegador.

```
String.prototype.capitalize = function()
{
  var firstLetter =
this[0].toUpperCase();
  var remainder =
Array.prototype.slice.call(this,
1).join('');
  return firstLetter + remainder;
};

"título".capitalize();
// "Título"
```

Patrones de diseño

Closure, Module, Callbacks, Publish/Subscribe

Scope

- Los *scopes* controlan la visibilidad de variables en un programa.
- Las variables definidas dentro de un *scope* están disponibles dentro del *scope* pero no fuera.

```
var UserAge = function(age) {
  return {
    sayAge: function() {
      var adult = false;
      if (age >= 18) { adult = true; }

      return age + ' years old'; // "23 years old"
    },
    isAdult: function() {
      return adult; // ReferenceError: adult is not
defined
    }
  };
};
```

Context

- El *context* es el “dueño” del *scope* más inmediato.
- Dentro de una función, puede ser llamado con `this`.

```
var UserAge = function(age) {
  return {
    adult: false,
    sayAge: function() {
      return age + ' years old';
    },
    isAdult: function() {
      this.adult = age >= 18;
      return this.adult;
    }
  };
};
```

Context y apply

- Con `call` y `apply` se pueden ejecutar funciones cambiando su *context*.
- Estos casos son útiles cuando se desea ejecutar una función en un *scope* diferente al que se encuentre.

```
var showContext = function() {
 console.log(this);
};

showContext();
// window
showContext.apply(1);
// 1
showContext.apply({});
// {}
```

Closure

- Un *closure* es una función definida dentro de otra función que tiene acceso al *scope* de la función que la contiene.
- En este caso, `sayHi` es un *closure* definido dentro de `UserName` y tiene acceso al *scope* de `UserName`.

```
var UserName = function(firstName, lastName) {
  var fullName = firstName + ' ' + lastName;

  var sayHi = function() {
    return 'Hello, my name is ' + fullName;
  };

  return sayHi();
};

UserName('Gustavo', 'Leon');
// Hello, my name is Gustavo Leon
```

Module

```
var users = [];

function add_user(user){
  users.push(user);
}

function remove_user(user) {
  var index = users.indexOf(user);
  users.splice(index, 1);
}

function count_users() {
  return users.length;
}
```

```
var UserModule = (function() {
  var users = [];

  return {
    add: function(user) {
      users.push(user);
    },
    remove: function(user) {
      users.splice(users.indexOf(user), 1);
    },
    count: function() {
      return users.length;
    }
  };
})();
```

Module – Scopes

```
// scope global
var UserModule = (function() {
  // scope de UserModule
  var users = [];
  return {
    add : function(user) {
      // scope de add
      users.push(user);
    },
    remove : function(user) {
      users.splice(users.indexOf(user), 1);
    }
  };
})();
```

Module - Contexts

```
var UserModule = (function() {
  var users = [];

  return {
    add : function(user) {
      users.push(user);
      alert(this.count() + " users added");
    },
    count : function() {
      return users.length;
    }
  };
})();
```

Callbacks

- Al ser las funciones First-Class Citizens, estas pueden ser pasadas como parámetros en otras funciones.
- Un *callback* es una función pasada como parámetro que es ejecutada después de determinado tiempo.

```
window.setTimeout(function(){
  console.log('Hello Vietnam!');
}, 1500);
```

Publish/Subscribe

```
var UserNotifier = (function() {
  var channels = {};

  return {
    publish : function(channel) {
      // Enviar un mensaje a todos los suscriptores de un canal
    },
    subscribe : function(channel, func) {
      // Suscribir a "channel" y le asignamos una acción
    },
    unsubscribe : function(id) {
      // Eliminar la suscripción con el id generado
    }
  };
})();
```